US008875064B2

(12) **United States Patent**
Aggarwal et al.

(10) **Patent No.: US 8,875,064 B2**
(45) **Date of Patent: Oct. 28, 2014**

(54) **AUTOMATED DESIGN RULE CHECKING (DRC) TEST CASE GENERATION**

(71) Applicant: **International Business Machines Corporation**, Armonk, NY (US)

(72) Inventors: **Davinder Aggarwal**, Bangalore (IN); **Vibhor Jain**, Bangalore (IN); **Janakiraman Viraraghavan**, Bangalore (IN)

(73) Assignee: **International Business Machines Corporation**, Armonk, NY (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **13/833,028**

(22) Filed: **Mar. 15, 2013**

(65) **Prior Publication Data**

US 2014/0282329 A1 Sep. 18, 2014

(51) **Int. Cl.**
G06F 17/50 (2006.01)

(52) **U.S. Cl.**
CPC ***G06F 17/5081*** (2013.01)
USPC **716/52**; 716/53; 716/54; 716/55; 716/106; 716/112

(58) **Field of Classification Search**
USPC 716/51–53, 106–115
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,724,504	A		3/1998	Aharon et al.
5,774,358	A		6/1998	Shrote
6,063,132	A	*	5/2000	DeCamp et al. 716/112
6,212,667	B1		4/2001	Geer et al.
6,526,546	B1	*	2/2003	Rolland et al. 324/762.06
6,606,735	B1	*	8/2003	Richardson et al. 716/112
6,611,946	B1	*	8/2003	Richardson et al. 716/112
6,732,338	B2		5/2004	Crouse et al.
7,254,791	B1	*	8/2007	Agrawal et al. 716/112
7,430,729	B2	*	9/2008	McLain et al. 716/100
7,757,190	B2	*	7/2010	Dai et al. 716/52
2006/0218516	A1	*	9/2006	McLain et al. 716/10
2007/0038970	A1	*	2/2007	DeMaris et al. 716/4
2009/0187867	A1	*	7/2009	Lawrence 716/4

OTHER PUBLICATIONS

"Design Rule Checking", Wikipedia; http://en.wikipedia.org/wiki/Design_rule_checking; last modified on Mar. 7, 2013, and previously modified on Nov. 17, 2012; 3 Pages.

* cited by examiner

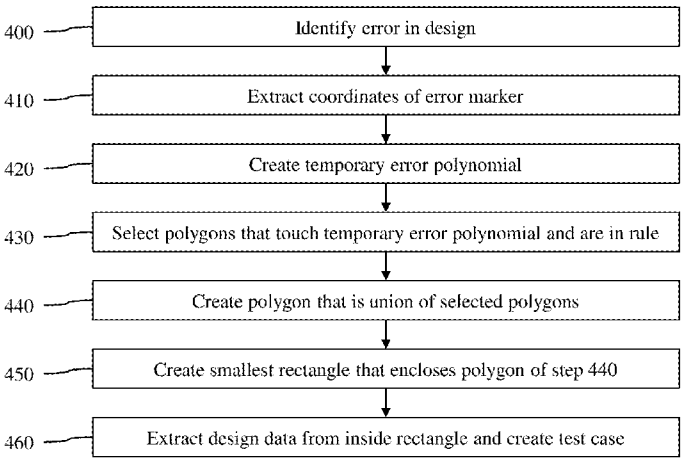
Primary Examiner — Nha Nguyen

(74) *Attorney, Agent, or Firm* — David Cain; Roberts Mlotkowski Safran & Cole, P.C.

(57) **ABSTRACT**

Approaches for generating test cases for design rule checking are provided. A method includes extracting coordinates of an error marker in an integrated circuit design. The method also includes creating an error polygon using the coordinates. The method additionally includes selecting polygons in the design that touch the error polygon. The method further includes identifying a rectangle that encloses the selected polygons. The method also includes generating a test case based on data of the design contained within the rectangle. The extracting, the creating, the selecting, the identifying, and the generating are performed using a computer device.

16 Claims, 7 Drawing Sheets



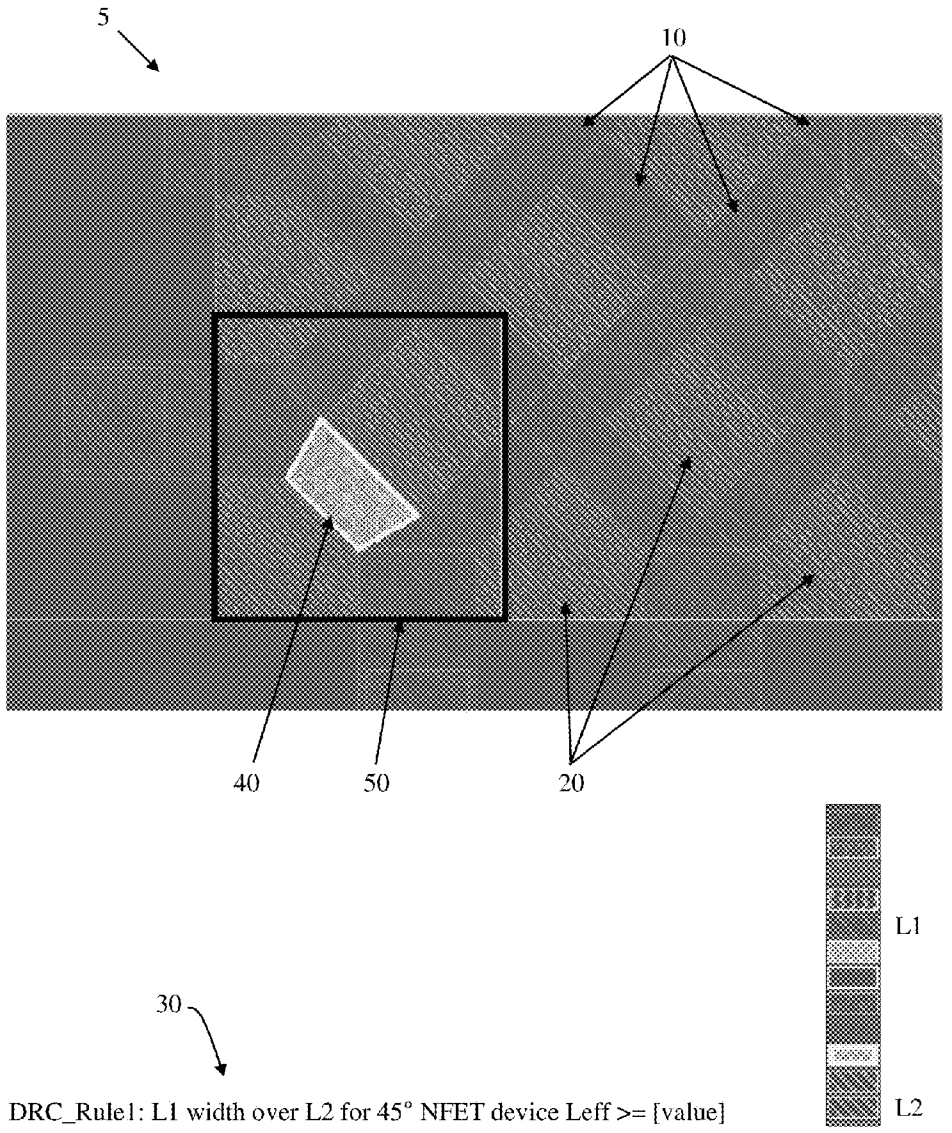

5
10
40
50
20
L1
L2
30
DRC_Rule1: L1 width over L2 for 45° NFET device Leff >= [value]

FIG. 1

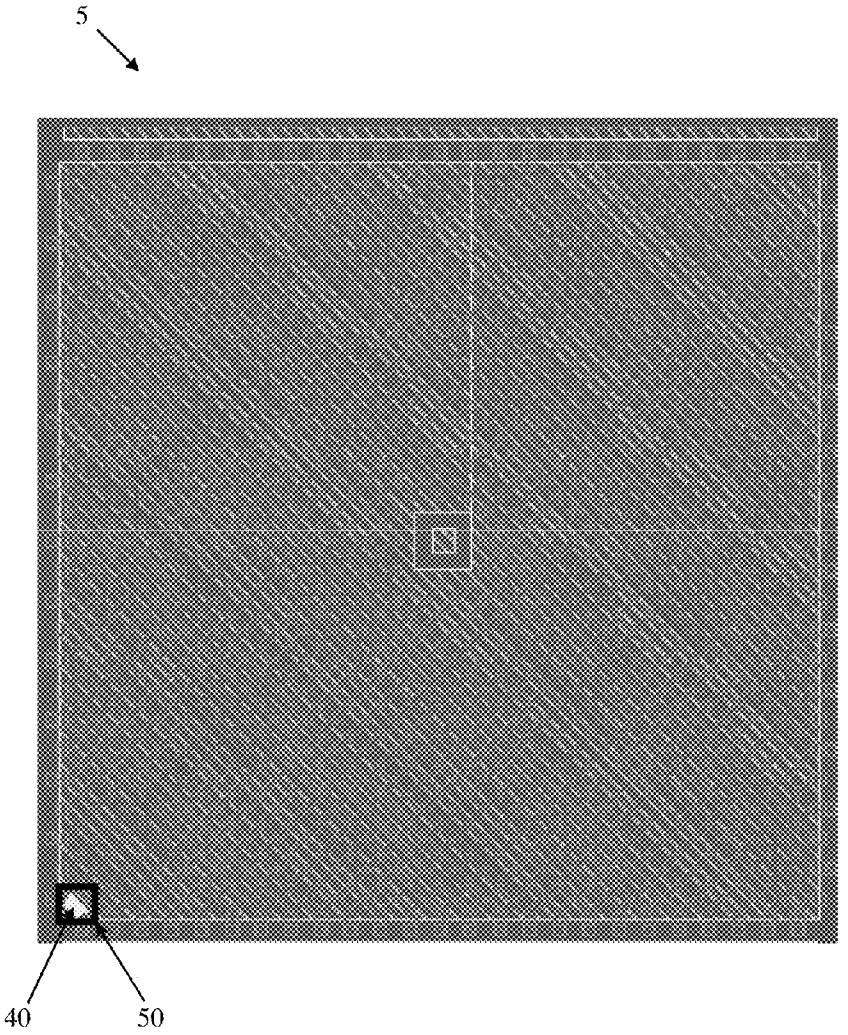


FIG. 2

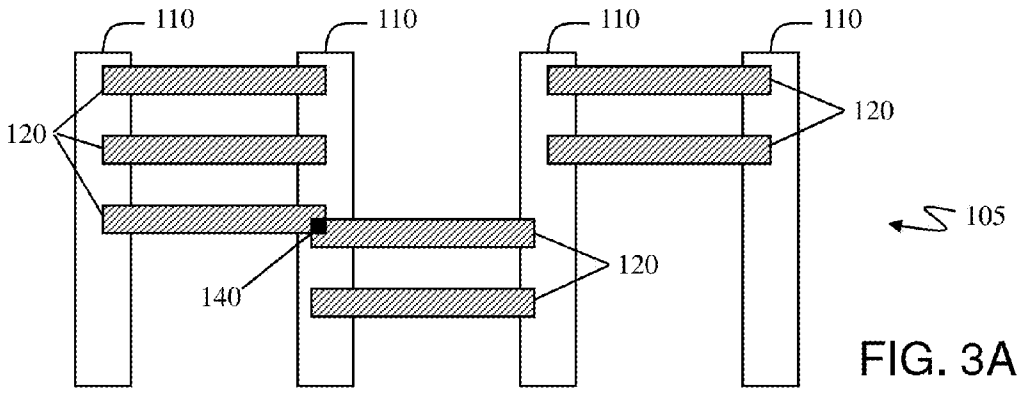


FIG. 3A

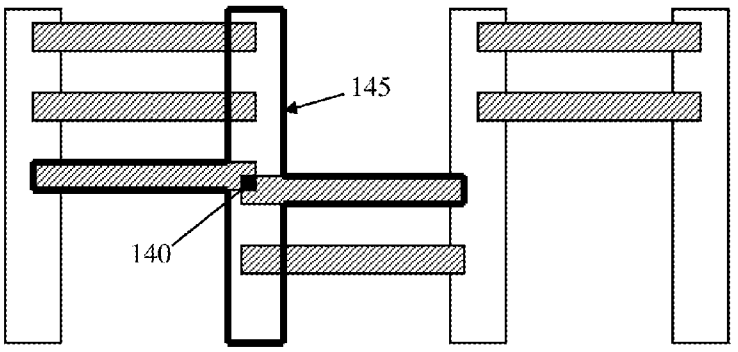


FIG. 3B

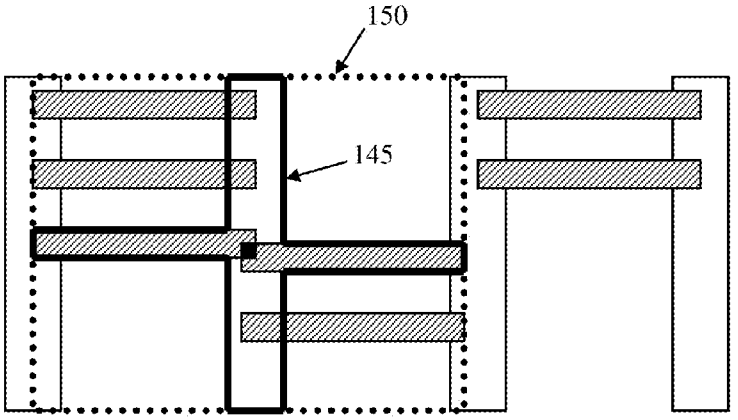


FIG. 3C

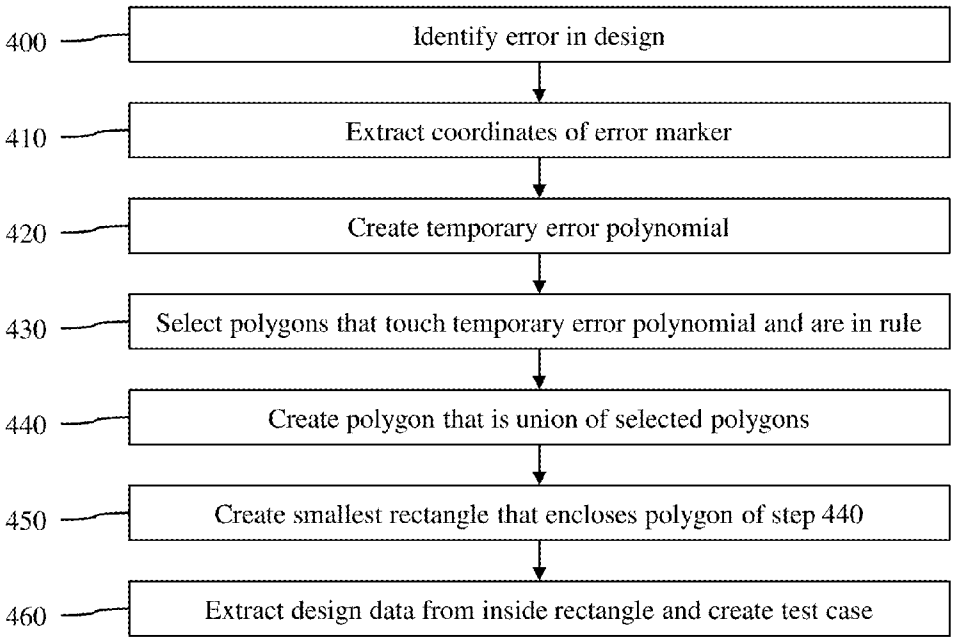

400
Identify error in design
410
Extract coordinates of error marker
420
Create temporary error polynomial
430
Select polygons that touch temporary error polynomial and are in rule
440
Create polygon that is union of selected polygons
450
Create smallest rectangle that encloses polygon of step 440
460
Extract design data from inside rectangle and create test case

FIG. 4

DRC_Rule1: L1 width over L2 for 45° NFET device Leff >= [value] ← 505

Layers involved in DRC_Rule1: L1, L2, L3, L4, L5, L6 ← 510

9839840 14314580
9840100 14314120
9840860 14314880
9840400 14315140 ← 515

520

DRC Code	Comment
INCLUDE $TECHDIR/DRC/Include/*.layers.cal	Layer definition extracted from original DRC runset
LAYER TEMP_ERROR 5953	Temporary error layer
POLYGON 9839.840000 14314.580000 9840.100000 14314.120000 9840.860000 14314.880000 9840.400000 14315.140000 TEMP_ERROR	Temporary error polynomial
TEST DRC_Rule1 {@ Generating test case for DRC_Rule1 COPY (EXTENT ((L1 OR (L2 OR (L3 OR (L4 OR (L5 OR L6))))) INTERACT TEMP_ERROR))}	INTERACT: selects polygons that touch the error polynomial EXTENT: smallest rectangle encloses polygons touching error polynomial

525

FIG. 5

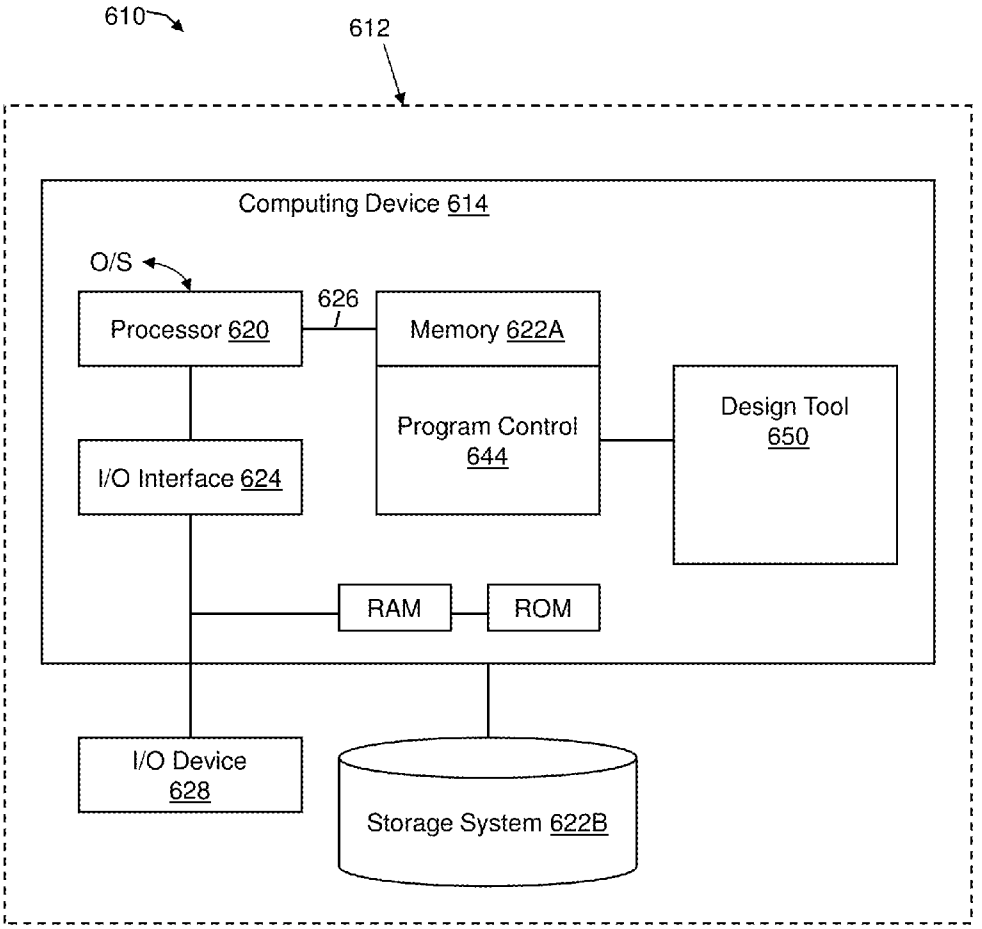

610
612
Computing Device 614
O/S
626
Processor 620
Memory 622A
Program Control 644
Design Tool 650
I/O Interface 624
RAM
ROM
I/O Device 628
Storage System 622B

FIG. 6

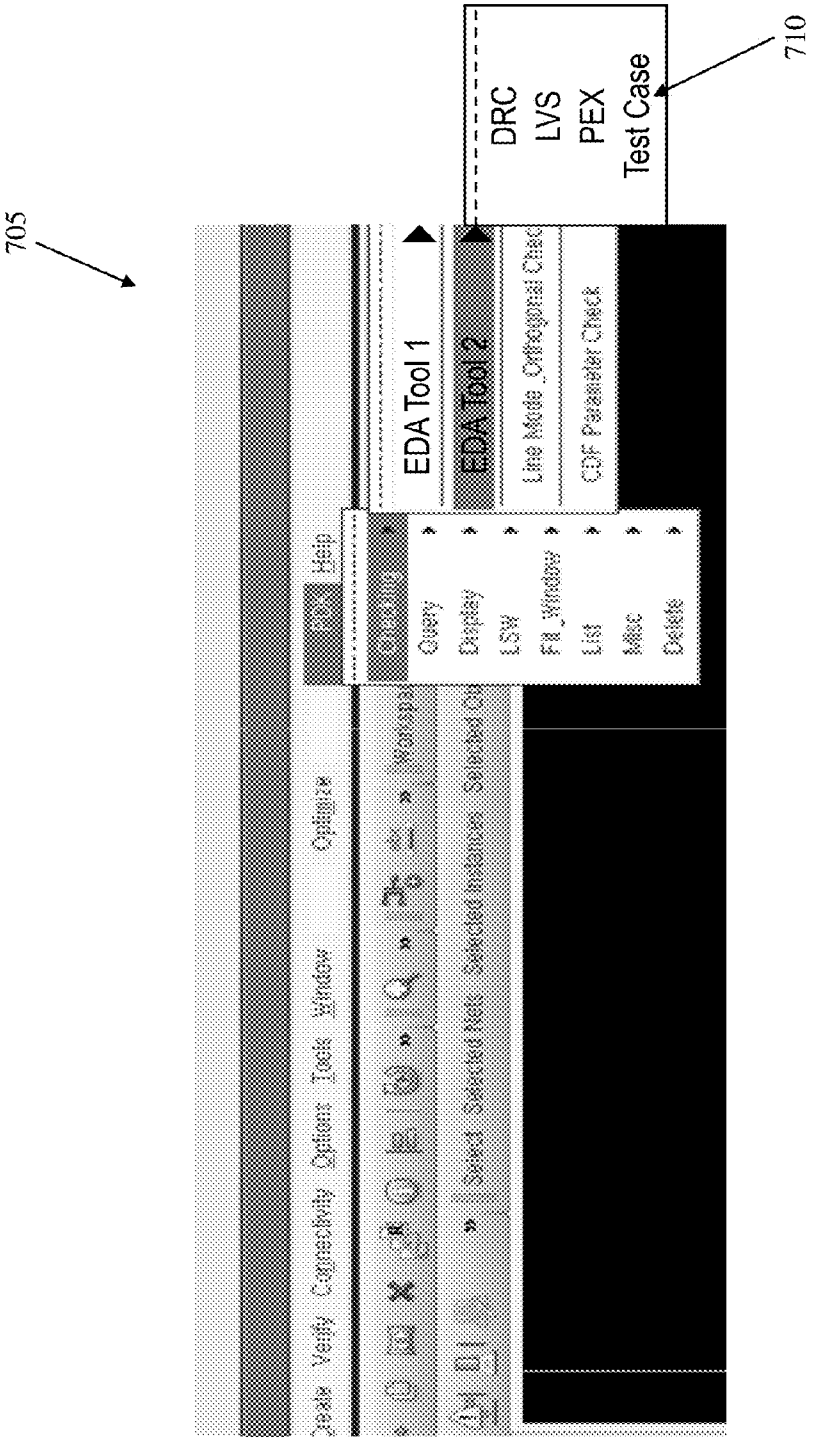

705
710
DRC
LVS
PEX
Test Case
EDA Tool 1
EDA Tool 2
Line Mode_Orthogonal Chec
CDF Parameter Check
Help
Create
Query
Display
LSW
Fit_Window
List
Misc
Delete
Verify
Connectivity
Options
Tools
Window
Optimize

FIG. 7

AUTOMATED DESIGN RULE CHECKING (DRC) TEST CASE GENERATION

FIELD OF THE INVENTION

The invention relates to testing of integrated circuits and, more particularly, to computer-aided testing for design verification of integrated circuits.

BACKGROUND

Design rule checking (DRC) is used in electronic design automation (EDA) of integrated circuits to determine whether the physical layout of a particular chip design satisfies a series of recommended parameters called design rules. Design rules are a series of parameters provided by semiconductor manufacturers that enable the designer to verify the correctness of a mask set. Design rules are specific to a particular semiconductor manufacturing process. A design rule set specifies certain geometric and connectivity restrictions to ensure sufficient margins to account for variability in semiconductor manufacturing processes, so as to ensure that most of the parts work correctly.

Specific design rule checks verify the shape and sizes of various circuit components that are diffused, deposited, or etched onto a chip. Additionally, design rule checking also verifies that the shapes are of the proper size, shape, and type, and furthermore, that the shapes are not placed so close together within the chip that they will not work. Design rule checking may involve a general purpose shapes processing program (GPSPP) that receives inputs from two files: runset and physical layout files. The runset file is a command language input file that instructs the processor executing the GPSPP how to perform the design rule checks. The runset may include several hundred individual design rule checks, for example. The runset may also be referred to as a DRC runset, a rule deck, or merely a deck.

Design rules (also referred to as DRC rules) specify how the layers in the layout should be arranged to ensure good manufacturing yield in a foundry. The runset is provided by the foundry and is coded based on a design manual. Inputs from the technology development and manufacturing teams, and information on the devices supported in a particular technology, are used by the design manual team to create the DRC rules in the design manual. It is thus advantageous to ensure that the runset is consistent with the design manual since customers are expected to ensure their designs are "DRC clean" on this "golden" runset.

When developing a runset for a semiconductor process, a set of layout test cases is used to verify functionality and accuracy. The task of creating test cases for runsets exists across all organizations and companies that code checking runsets. The code for DRC is created based on a set of layout design rules or parameters for a particular semiconductor process. The code and test cases are both manually created.

A runset may be validated with regression testing that uses shape-based test cases that are based on rules described in the design manual. For example, regression testing involves creating such test cases and verifying the test cases against the runset. The test cases used in regression testing are not based on an actual circuit design, but rather are simple shapes based on rules included in the design manual and designed to trigger either a pass condition or a fail condition when verified against the runset. Both pass test cases and fail test cases are built to ensure good verification coverage of the design rules. For example, the fail test cases are designed to cause the runset to report an error, and the pass test cases are designed such that the runset should not report an error. In the event the runset does not behave as expected according to the test cases, then one or more design rule checks in the runset may be modified, or the design manual itself may be modified, or both.

Since the test cases used in regression testing are manually created, they are necessarily limited by the imagination and/or expertise of the person tasked with creating the test cases. This person-based limitation can limit the verification coverage provided by the test cases. The verification coverage is also limited since the number of ways in which a rule can be violated grows exponentially with the number of layers/constraints involved in the rule. As the number of layers and constraints in a design increases, it becomes unworkable to manually create test cases that provide sufficient verification coverage.

SUMMARY

In a first aspect of the invention, there is a method of generating a test case in design rule checking. The method includes extracting coordinates of an error marker in an integrated circuit design. The method also includes creating an error polygon using the coordinates. The method additionally includes selecting polygons in the design that touch the error polygon. The method further includes identifying a rectangle that encloses the selected polygons. The method also includes generating a test case based on data of the design contained within the rectangle. The extracting, the creating, the selecting, the identifying, and the generating are performed using a computer device.

In another aspect of the invention, there is a system for generating a test case. The system includes a computer device including a processor and a design tool that is structured and arranged to: perform design rule checking of an integrated circuit design; identify a violation of a design rule during the design rule checking; extract coordinates of an error marker associated with the violation; create an error polygon using the coordinates; select polygons in the design that touch the error polygon; identify a rectangle that encloses the selected polygons; and generate a test case based on data of the design contained within the rectangle.

In another aspect of the invention, there is a computer program product for generating a test case. The computer program product comprises a computer readable storage medium having program code embodied therewith, the program code being readable and/or executable by a processor of a computer device to perform a method. The method includes: performing, by the processor, indentifying, by the processor, a violation of a design rule during design rule checking of an integrated circuit design; creating, by the processor, an error polygon based on the violation; identifying, by the processor, a rectangle that encloses polygons in the design that are associated with the design rule and that touch the error polygon; and generating, by the processor, a test case based on data of the design contained within the rectangle.

BRIEF DESCRIPTION OF THE SEVERAL VIEWS OF THE DRAWINGS

The present invention is described in the detailed description which follows, in reference to the noted plurality of drawings by way of non-limiting examples of exemplary embodiments of the present invention.

FIG. **1** depicts a portion of an integrated circuit design in accordance with aspects of the invention;

FIG. **2** depicts a portion of an integrated circuit design in accordance with aspects of the invention;

FIGS. **3**A-**3**C depict steps of generating an area for extracting test case data in accordance with aspects of the invention;

FIG. **4** shows a flowchart of a process in accordance with aspects of the invention;

FIG. **5** shows exemplary programming for performing aspects of the invention;

FIG. **6** shows an illustrative environment for performing the processes in accordance with the invention; and

FIG. **7** shows an exemplary interface providing functionality in accordance with aspects of the invention.

DETAILED DESCRIPTION

The invention relates to testing of integrated circuits and, more particularly, to computer-aided testing for design verification of integrated circuits. According to aspects of the invention, a test case for verifying a DRC runset is automatically generated based on an error that is identified when performing design rule checking of an existing integrated circuit design. In embodiments, the error is identified by performing design rule checking of the integrated circuit design using two different design tools that use two different runsets, and identifying discrepancies between results of the two different design tools. In implementations, the test case is automatically generated by: extracting coordinates of an error marker associated with the error; creating a polygon that is the union of all shapes in the integrated circuit design that touch the error marker; determining the coordinates of a smallest rectangle that encloses the polygon; and generating a unit level test case based on the coordinates of a smallest rectangle.

In this manner, test cases for verifying DRC runsets may be automatically generated based on actual integrated circuit designs. Moreover, by including only the shapes that touch the error region, the automatically generated test cases in accordance with aspects of the invention contain sufficient information for debugging the runset, but are not so large as to impose great computational penalties during the runset debugging.

Test cases generated in aspects of the invention may be used to verify the runset of a design tool, in addition to regression testing. Test cases generated in aspects of the invention improve test coverage of the runset, compared to regression testing alone, by looking at failures that occur in an existing integrated circuit design, e.g., at the highest level in a hierarchy of a larger integrated circuit design.

Testing of an existing integrated circuit design (referred to as "IP testing") addresses the issue of testing the runset on complicated yet practical test cases. In IP testing, the runset is verified on an existing integrated circuit design that is typically very large. The existing design may include a customer's design of an integrated circuit, for example, which means that test cases generated from IP testing have a practical basis and thus are well-suited for verifying a DRC runset. Any errors reported on existing integrated circuit designs are necessarily true errors, which should not be missed. False errors and missed errors are detected in aspects of the invention by performing DRC testing of the existing design using two different tools having two different runsets, and noting any discrepancy of when one of the tools indicates an error that the other tool does not. Such a discrepancy can be due to either a false error (i.e., the design does not have an error, meaning that the tool reporting the error has a bug) or a missed error (i.e., the design does have an error, meaning that the tool not reporting the error has a bug). A true error is where both design tools agree in indicating an error in the design, in which case there is not a discrepancy between the design tools and no test case is generated for such an error.

It is noted that a discrepancy is not always due to a bug in one of the runsets that results in a change being made to one of the runsets. For example, discovering such a discrepancy may result in changing design manual wording or reporting a tool limitation to the DRC tool vendor. Specifically, although the design manual specifies a particular intent of a DRC rule, the DRC rule when written and implemented may not capture and/or convey the intent. In such cases, finding of a discrepancy during IP testing may result in changing the wording of the design manual rather than changing the DRC code.

IP testing may be performed on large existing designs to the extent that the design is even able to be loaded in a DRC tool. Due to the large size of existing designs, viewing a DRC error in the DRC tool can be difficult. It is therefore desirable to extract relatively small test cases from the larger design, which test cases are representative of the actual error found in the design. To this end, a representative test case should: be much smaller than the whole existing design; reproduce the error seen in the existing design; and clearly show the structure that resides in the design. Showing the structure that resides in the design is useful since a design manual change may result from the discrepancy, in which case it is useful to be aware of the exact structure in the design to determine whether it is consistent with the intent of the design manual.

FIG. **1** shows an example of an error in an existing integrated circuit design **5** in IP testing in accordance with aspects of the invention. The exemplary design includes shapes **10** (e.g., features, objects, etc.) formed in a first layer "L1" and shapes **20** (e.g., features, objects, etc.) formed in a second layer "L2" of the integrated circuit. An exemplary DRC rule **30** is also shown in FIG. **1**. The exemplary rule **30** specifies a width of L1 over L2 at 45°. An error marker **40** shows a location of an error, i.e., a violation of the rule **30**.

The display of FIG. **1** may be generated and displayed using a DRC tool, e.g., by loading a runset including at least DRC rule **30** into the tool, loading a design including at least L1 and L2 into the tool, and verifying the design against the runset using the tool. The DRC tool may be implemented, for example, using special purpose programming that is loaded in and executed on a computer device, e.g., as described with respect to FIG. **6** herein. The DRC tool may be a standalone program, or included as a module of another program such as a process design kit (PDK), e.g., as described with respect to FIG. **7** herein.

Still referring to FIG. **1**, one approach to generating a test case based on the error marker **40** is to draw a rectangle **50** around the error marker **40**, and create a test case data structure (e.g., a file) that includes a portion of the design data included within the bounds of the rectangle **50**. The format of the data can be any suitable format, such as graphic data system (GDS), GDSII, etc. Manually drawing the rectangle **50** involves some action (e.g., input) by a user of the DRC tool, and thus does not represent a fully automated approach for generating the test case. Moreover, arbitrarily selecting a size of the rectangle **50** does not necessarily include sufficient structure of the design **5** to permit an engineer or designer to determine whether the design as a whole complies with the intent of the design manual, e.g., when debugging the runset using the generated test case.

FIG. **2** shows a larger portion of the design **5** of FIG. **1**. As depicted in FIG. **2**, the design **5** includes an allowable valid structure that is not discernable when only viewing the rectangle **50** surrounding error marker **40**. An exemplary solution to this problem is to ask the designer to add a waiver layer to

prevent this error from being flagged on the structure, in which case no change in the DRC deck is required.

FIGS. **3**A-**3**C depict steps of generating an area for extracting test case data in accordance with aspects of the invention. In particular, FIG. **3**A shows an exemplary portion of another design **105** including shapes **110** in a first layer and shapes **120** in a second layer. Error marker **140** represents the location of an error where the design **105** violates a DRC rule contained in a runset.

With reference to FIGS. **3**B and **3**C, and according to aspects of the invention, a test case is created by: identifying all the shapes that touch the error marker **140**; creating a polygon **145** that is a union of all the shapes touching the error marker **140** (FIG. **3**B); creating the smallest rectangle **150** that encloses the polygon **145** (FIG. **3**C); and creating a test case data structure (e.g., a file) by extracting and saving data of the design **105** (e.g., GDS data) that is included in the rectangle **150**. In embodiments, the smallest rectangle **150** is the rectangle that is coincident with outermost edges of the polygon **145**. By using the smallest rectangle, implementations of the invention generate a test case that is relatively small so as not to be too computationally expensive during debugging, but which test case still contains enough design information (e.g., the shapes contained in the rectangle) to provide the debugger with context of the design surround the error.

In embodiments, identifying the shapes, creating the polygon, creating the smallest rectangle **150**, and creating the test case data structure are all performed automatically. For example, a script (e.g., routine, program, etc.) may be written that uses commands of the DRC tool to perform these steps in this order, thereby resulting in the automatic generation of a test case associated with the error. For example, the script (e.g., routine, program, etc.) may utilize commands such as: "interact" to select shapes (e.g., polygons) that touch the error marker **140**; "extent" to create the smallest rectangle **150**; and "yank" or "layout copy" to extract GDS data of the design that is included in the rectangle **150**.

FIG. **4** shows an exemplary flowchart and/or block diagram for performing aspects of the present invention. The steps of FIG. **4** may be implemented, for example, in the environment of FIG. **6**, which is described in greater detail herein.

The flowcharts and block diagrams in the Figures illustrate the architecture, functionality, and operation of possible implementations of systems, methods and computer program products according to various embodiments of the present invention. In this regard, each block in the flowchart or block diagrams may represent a module, segment, or portion of code, which comprises one or more executable instructions for implementing the specified logical function(s). It should also be noted that, in some alternative implementations, the functions noted in the block may occur out of the order noted in the figures. For example, two blocks shown in succession may, in fact, be executed substantially concurrently, or the blocks may sometimes be executed in the reverse order, depending upon the functionality involved. It will also be noted that each block of the block diagrams and/or flowchart illustration, and combinations of blocks in the block diagrams and/or flowchart illustration, can be implemented by special purpose hardware-based systems that perform the specified functions or acts, or combinations of special purpose hardware and computer instructions.

Furthermore, the invention can take the form of a computer program product accessible from a computer-usable or computer-readable medium providing program code for use by or in connection with a computer or any instruction execution system. The software and/or computer program product can be implemented in the environment of FIG. **6**.

FIG. **4** depicts an exemplary flowchart for a process in accordance with aspects of the present invention. The steps of FIG. **4** may be performed by a computer-based design tool using commands that are included in (e.g., programmed in) the design tool. The computer-based design tool may be implemented in the environment of FIG. **6**, for example, and may comprise a combination of hardware and special-purpose software that is configured to perform the functions described herein.

At step **400**, the design tool identifies an error (e.g., a design rule violation) in a design of an integrated circuit by applying at least one DRC rule included in a runset against the integrated circuit design. The error represents a situation where the design violates the DRC rule. Step **400** may include loading layout data (e.g., GDS data) of the design into the design tool. Step **400** may also include loading two different runsets into the design tool. In embodiments, step **400** may further include verifying the design using the two different runsets and identifying any false errors or missed errors. In this case, the error identified at step **400** is an error that is reported by only one of the two runsets, which indicates a discrepancy between the runsets. Alternatively, the error identified at step **400** may be determined by verifying the design using a single runset, in which case it is not determined whether the error is a false error, a missed error, or a true error.

At step **410**, the design tool extracts the coordinates of an error marker for the error identified at step **400**. At step **420**, the design tool creates a temporary polygon layer with the error coordinates from step **410**. The temporary polygon layer may be referred to as the temporary error polygon (or error shape). At step **430**, the design tool identifies (e.g., selects) a set of polygons in the design that touch the error polygon and are included in a list of layers associated with the DRC rule that is the basis of the error from step **400**. At optional step **440**, the design tool creates a polygon which is the union of the set of polygons identified at step **430**. At step **450**, the design tool identifies a smallest rectangle that encloses the polygon created at step **440** or, when step **440** is omitted, the smallest rectangle that contains all of the polygons touching the temporary error polygon as identified at step **530**. At step **460**, the design tool extracts data from the design data that is included within the rectangle identified at step **450** and saves the data in a test case data structure.

FIG. **5** shows an exemplary DRC rule **505** which may represent, for example, the DRC rule that generated the error at step **400** of FIG. **4**. In embodiments, the DRC rule **505** is associated with a list of layers **510** of the integrated circuit design. FIG. **5** also depicts exemplary coordinates **515** of the error marker determined at step **410**. FIG. **5** further depicts program code for implementing function described with respect to the flowchart of FIG. **4**. For example, code portions **520** may be employed to implement the functionality of step **420**, and code portion **525** may be employed to implement the functionality of steps **430**, **440**, and **450**. Although not shown, a command such as "yank", "layout copy", or the like may be employed to implement the functionality of step **460**. The code portions **520** are implemented in CALIBRE®, which is a trademark of Mentor Graphics Corporation of Wilsonville, Oreg. Aspects of the invention may be implemented using any suitable computer design software for electronic design automation, and are not limited to the particular software shown in FIG. **5**.

FIG. **6** shows an illustrative environment **610** for managing the processes in accordance with the invention. As will be appreciated by one skilled in the art, aspects of the present

invention may be embodied as a system, method, or computer program product. Accordingly, aspects of the present invention may take the form of an entirely hardware embodiment, an entirely software embodiment (including firmware, resident software, micro-code, etc.) or an embodiment combining software and hardware aspects that may all generally be referred to herein as a "circuit," "module" or "system." Furthermore, aspects of the present invention may take the form of a computer program product embodied in one or more computer readable medium(s) having computer readable program code embodied thereon.

Any combination of one or more computer readable medium(s) may be utilized. The computer readable medium may be a computer readable signal medium or a computer readable storage medium. A computer readable storage medium may be, for example, but not limited to, an electronic, magnetic, optical, electromagnetic, infrared, or semiconductor system, apparatus, or device, or any suitable combination of the foregoing. More specific examples (a non-exhaustive list) of the computer readable storage medium would include the following: an electrical connection having one or more wires, a portable computer diskette, a hard disk, a random access memory (RAM), a read-only memory (ROM), an erasable programmable read-only memory (EPROM or Flash memory), an optical fiber, a portable compact disc read-only memory (CD-ROM), an optical storage device, a magnetic storage device, or any suitable combination of the foregoing. In the context of this document, a computer readable storage medium may be any tangible medium that can contain, or store a program for use by or in connection with an instruction execution system, apparatus, or device.

A computer readable signal medium may include a propagated data signal with computer readable program code embodied therein, for example, in baseband or as part of a carrier wave. Such a propagated signal may take any of a variety of forms, including, but not limited to, electro-magnetic, optical, or any suitable combination thereof. A computer readable signal medium may be any computer readable medium that is not a computer readable storage medium and that can communicate, propagate, or transport a program for use by or in connection with an instruction execution system, apparatus, or device.

Program code embodied on a computer readable medium may be transmitted using any appropriate medium, including but not limited to wireless, wireline, optical fiber cable, RF, etc., or any suitable combination of the foregoing.

Computer program code for carrying out operations for aspects of the present invention may be written in any combination of one or more programming languages, including an object oriented programming language such as Java, Smalltalk, C++ or the like and conventional procedural programming languages, such as the "C" programming language or similar programming languages. The program code may execute entirely on the user's computer, partly on the user's computer, as a stand-alone software package, partly on the user's computer and partly on a remote computer or entirely on the remote computer or server. In the latter scenario, the remote computer may be connected to the user's computer through any type of network, including a local area network (LAN) or a wide area network (WAN), or the connection may be made to an external computer (for example, through the Internet using an Internet Service Provider).

Aspects of the present invention are described below with reference to flowchart illustrations and/or block diagrams of methods, apparatus (systems) and computer program products according to embodiments of the invention. It will be understood that each block of the flowchart illustrations and/or block diagrams, and combinations of blocks in the flowchart illustrations and/or block diagrams, can be implemented by computer program instructions. These computer program instructions may be provided to a processor of a general purpose computer, special purpose computer, or other programmable data processing apparatus to produce a machine, such that the instructions, which execute via the processor of the computer or other programmable data processing apparatus, create means for implementing the functions/acts specified in the flowchart and/or block diagram block or blocks.

These computer program instructions may also be stored in a computer readable medium that can direct a computer, other programmable data processing apparatus, or other devices to function in a particular manner, such that the instructions stored in the computer readable medium produce an article of manufacture including instructions which implement the function/act specified in the flowchart and/or block diagram block or blocks.

The computer program instructions may also be loaded onto a computer, other programmable data processing apparatus, or other devices to cause a series of operational steps to be performed on the computer, other programmable apparatus or other devices to produce a computer implemented process such that the instructions which execute on the computer or other programmable apparatus provide processes for implementing the functions/acts specified in the flowchart and/or block diagram block or blocks.

Still referring to FIG. **6**, the environment **610** includes a server or other computing system **612** that can perform the processes described herein. In particular, the system **612** includes a computing device **614**. The computing device **614** can be resident on a network infrastructure or computing device of a third party service provider (any of which is generally represented in FIG. **6**).

The computing device **614** also includes a processor **620**, memory **622**A, an I/O interface **624**, and a bus **626**. The memory **622**A can include local memory employed during actual execution of program code, bulk storage, and cache memories which provide temporary storage of at least some program code in order to reduce the number of times code must be retrieved from bulk storage during execution. In addition, the computing device includes random access memory (RAM), a read-only memory (ROM), and an operating system (O/S).

The computing device **614** is in communication with the external I/O device/resource **628** and the storage system **622**B. For example, the I/O device **628** can comprise any device that enables an individual to interact with the computing device **614** (e.g., user interface) or any device that enables the computing device **614** to communicate with one or more other computing devices using any type of communications link. The external I/O device/resource **628** may be for example, a handheld device, PDA, handset, keyboard etc.

In general, the processor **620** executes computer program code (e.g., program control **644**), which can be stored in the memory **622**A and/or storage system **622**B. Moreover, in accordance with aspects of the invention, the program control **644** controls a design tool **650**, e.g., that performs one or more of the processes described herein. The design tool **650** can be implemented as one or more program code in the program control **44** stored in memory **622**A as separate or combined modules. Additionally, the design tool **650** may be implemented as separate dedicated processors or a single or several processors to provide the function of these tools. While executing the computer program code, the processor **620** can read and/or write data to/from memory **622**A, storage system

622B, and/or I/O interface **624**. The program code executes the processes of the invention. The bus **626** provides a communications link between each of the components in the computing device **614**.

The computing device **614** can comprise any general purpose computing article of manufacture capable of executing computer program code installed thereon (e.g., a personal computer, server, etc.). However, it is understood that the computing device **614** is only representative of various possible equivalent-computing devices that may perform the processes described herein. To this extent, in embodiments, the functionality provided by the computing device **614** can be implemented by a computing article of manufacture that includes any combination of general and/or specific purpose hardware and/or computer program code. In each embodiment, the program code and hardware can be created using standard programming and engineering techniques, respectively.

Similarly, the system **612** is only illustrative of various types of computer infrastructures for implementing the invention. For example, in embodiments, the system **612** comprises two or more computing devices (e.g., a server cluster) that communicate over any type of communications link, such as a network, a shared memory, or the like, to perform the process described herein. Further, while performing the processes described herein, one or more computing devices on the system **612** can communicate with one or more other computing devices external to the system **612** using any type of communications link. The communications link can comprise any combination of wired and/or wireless links; any combination of one or more types of networks (e.g., the Internet, a wide area network, a local area network, a virtual private network, etc.); and/or utilize any combination of transmission techniques and protocols.

FIG. **7** shows an exemplary computer-based graphical user interface **705** in accordance with aspects of the invention. As depicted in FIG. **7**, design tool software included in a process design kit (PDK) may be programmed to include a "Test Case" menu option **710** that generates a test case in a manner described herein, e.g., as described with respect to FIGS. **3**A-C, **4**, and **5**. In this manner, a customer may generate a test case and include the test case as part of a foundry change request (FCR) that is submitted to a foundry. Since the test case according to aspects of the invention does not include the entire integrated circuit design, the customer is able to avoid submitting the entire design to the foundry. This is advantageous for customers who are prohibited from sharing the entire design with a third party.

As described herein, implementations of the invention may be used to save significant amounts of time by generating test cases at the time of IP testing. Moreover, aspects of the methods described herein are technology independent and can be implemented using commands of commercial DRC tools. Test cases according to aspects of the invention may also be automatically generated at regression testing time and used to improve the debugging during regression testing. Furthermore, application engineers may use aspects of the invention to report DRC discrepancies and to reduce the size of test cases included in foundry change requests, e.g., from 36 Mb to 16 Kb in one example, and from 236 Mb to 246 Kb in another example.

As described herein, and according to aspects of the invention, when a discrepancy is seen between two DRC tools, the one flagging the error will display an error marker, e.g., a polygon, showing the region where the DRC rule is failing. Implementations of the invention identify a minimal region around the error marker which will reproduce the error. Arbitrarily choosing a large rectangular region around the error marker might reproduce the error but the test case might be too large, e.g., too computationally expensive. Conversely, a small rectangular region may not even reproduce the discrepancy. Implementations of the invention identify the smallest region around the error marker which reproduces the discrepancy by using the error marker as the reference. All shapes in the design involved in that error (e.g., that interact, touch, overlap, etc., the error marker) are used to generate a polygonal region which is involved in flagging this error. The coordinates of smallest rectangle enclosing this polygon are then extracted, which is in turn used to generate the unit level test case. The unit level test cases, owing to their small sizes, are relatively simpler than entire designs in analyzing and resolving the discrepancy. Additionally, these unit level test cases can be added to regression test libraries for future DRC deck validation.

The method as described above is used in the fabrication of integrated circuit chips. The resulting integrated circuit chips can be distributed by the fabricator in raw wafer form (that is, as a single wafer that has multiple unpackaged chips), as a bare die, or in a packaged form. In the latter case the chip is mounted in a single chip package (such as a plastic carrier, with leads that are affixed to a motherboard or other higher level carrier) or in a multichip package (such as a ceramic carrier that has either or both surface interconnections or buried interconnections). In any case, the chip is then integrated with other chips, discrete circuit elements, and/or other signal processing devices as part of either (a) an intermediate product, such as a motherboard, or (b) an end product. The end product can be any product that includes integrated circuit chips, ranging from toys and other low-end applications to advanced computer products having a display, a keyboard or other input device, and a central processor.

The descriptions of the various embodiments of the present invention have been presented for purposes of illustration, but are not intended to be exhaustive or limited to the embodiments disclosed. Many modifications and variations will be apparent to those of ordinary skill in the art without departing from the scope and spirit of the described embodiments. The terminology used herein was chosen to best explain the principles of the embodiments, the practical application or technical improvement over technologies found in the marketplace, or to enable others of ordinary skill in the art to understand the embodiments disclosed herein.

What is claimed is:

1. A method of generating a test case in design rule checking, comprising:

extracting coordinates of an error marker in an integrated circuit design;

creating an error polygon using the coordinates;

selecting polygons in the design that touch the error polygon;

identifying a rectangle that encloses the selected polygons;

generating a test case based on data of the design contained within the rectangle, wherein the extracting, the creating, the selecting, the identifying, and the generating are performed using a computer device;

identifying an error in the integrated circuit design, wherein:

the error indicates a violation of a design rule; and

the error maker is based on the error; and

wherein:

the identifying the error comprises testing the design using a first runset and a second runset; and

the error is a design rule violation of only one of the first runset and the second runset.

2. The method of claim 1, wherein the selecting the polygons comprises selecting the polygons in the design that touch the error polygon and which are included in a list of layers associated with a design rule.

3. The method of claim 2, wherein the error marker is based on a violation of the design rule.

4. The method of claim 1, wherein the rectangle is a smallest rectangle that encloses the selected polygons.

5. The method of claim 1, further comprising creating a polygon that is a union of the selected polygons.

6. The method of claim 5, wherein the rectangle is a smallest rectangle that encloses the created polygon.

7. The method of claim 1, wherein the generating the test case comprises extracting the data from the design.

8. The method of claim 7, wherein the generating the test case comprises saving the extracted data in a data structure.

9. The method of claim 1, further comprising using the test case to debug a runset.

10. The method of claim 1, further comprising automatically performing the extracting, the creating, the selecting, the identifying, and the generating.

11. A system for generating a test case, comprising:

a computer device comprising a processor and a design tool that is structured and arranged to:

perform design rule checking of an integrated circuit design;

identify a violation of a design rule during the design rule checking;

extract coordinates of an error marker associated with the violation;

create an error polygon using the coordinates;

select polygons in the design that touch the error polygon;

identify a rectangle that encloses the selected polygons;

generate a test case based on data of the design contained within the rectangle; and

wherein:

the performing design rule checking comprises testing the design using a first runset and a second runset; and

the violation is a design rule violation of only one of the first runset and the second runset.

12. The system of claim 11, wherein the generating the test case comprises:

extracting the data from the design; and

saving the extracted data in a test case data structure.

13. The system of claim 11, wherein the rectangle is a smallest rectangle that encloses the selected polygons.

14. The system of claim 11, wherein:

the design tool is included in a process design kit and includes a menu option for performing the generating the test case;

the rectangle is a smallest rectangle that encloses the selected polygons; and

the smallest rectangle is coincident with outermost edges of the selected polygons.

15. A computer program product for generating a test case, the computer program product comprising a computer readable storage medium having program code embodied therewith, the program code being readable and/or executable by a processor of a computer device to perform a method comprising:

identifying, by the processor, a violation of a design rule during design rule checking of an integrated circuit design;

creating, by the processor, an error polygon based on the violation;

identifying, by the processor, a rectangle that encloses polygons in the design that are associated with the design rule and that touch the error polygon; and

generating, by the processor, a test case based on data of the design contained within the rectangle,

wherein the rectangle is a smallest rectangle that encloses polygons in the design that are associated with the design rule and that touch the error polygon; and

wherein:

the design rule checking comprises testing the design using a first runset and a second runset; and

the violation is a design rule violation of only one of the first runset and the second runset.

16. The computer program product of claim 15, wherein the creating the error polygon and the identifying the rectangle are performed using commands in a design rule checking tool.

* * * * *